Peter Mileff PhD

SOFTWARE ENGINEERING

Agile Software Development

University of Miskolc
Department of Information Technology

Introduction…

Introduction

- Today businesses operate in a global, rapidly changing environment
- They have to respond to
  - new opportunities and markets,
  - changing economic conditions,
  - and the emergence of competing products and services
- **Software is part of almost all business operations**
  - new software is developed quickly to take advantage of new opportunities
    - and to respond to competitive pressure.
  - **Rapid development and delivery** is therefore often the most critical requirement for software systems.

Introduction

- Because of the rapidly changing environment
  - it is often practically impossible to derive a complete set of stable software requirements
  - The initial requirements inevitably change
    - customers find it impossible to predict how a system will affect working practices
    - how it will interact with other systems
    - and what user operations should be automated
- All of these caused the development processes that focus on **rapid software development and deliver**

Introduction

- Rapid software development processes are designed to produce useful software quickly.

- The software is not developed as a single unit
  - but as a series of increments,
  - each increment includes new system functionality

Rapid software development characteristics

- **The processes of specification, design, and implementation are interleaved:**
  - There is no detailed system specification
  - Design documentation is minimized or generated automatically by the programming environment
  - The user requirements document only defines the most important characteristics of the system.

Rapid software development characteristics

- **The system is developed in a series of versions:**
  - End-users and system stakeholders are involved in specifying and evaluating each version.
  - They may propose changes to the software and new requirements
    - that should be implemented in a later version of the system.

Rapid software development characteristics

- **System user interfaces are often developed using an interactive development system:**
  - that allows the interface design to be quickly created by drawing and placing icons on the interface.
  - The system may then generate
    - a web-based interface for a browser
    - or an interface for a specific platform such as Microsoft Windows or Linux.

Agile software development...

Agile development

- **In the 1980s and early 1990s:**
  - Typically plan driven development approaches
  - Principle: the best way to achieve better software was through
    - careful project planning,
    - formalized quality assurance,
    - rigorous software development processes
- Disadvantage for small and medium-sized business systems
  - Significant overhead in
    - planning, designing and system documentation
  - More time is spent on requirement analysis and design
    - rather than on program development and testing

Agile development

- In the 1990s, software developers propose new "**agile methods**".
- The main goal was
  - The development team should focus on the software itself
    - rather than on its design and documentation.

What is Agile Development?

Agile Development is an approach, a family of methodologies and not a particular software development method.

Agile manifesto

- The philosophy behind agile methods is reflected in the **Agile Manifesto**
  - Published in 2001 by many of the leading developers

This manifesto states:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile Development

- Agile software development is a **theoretical framework**
- There are many Agile software development processes
  - most of all seeks
    - to achieve a low-risk software development
    - width short development cycles (iterations)
- Every iteration is a complete software development cycle:
  - requirements analysis
  - design
  - implementation
  - testing and documentation
- The goal of an iteration is to perform
  - a tested and workable release of the application at the end of the cycle.

Agile Development

- Agile software development is a **theoretical framework**
- There are many Agile software development processes
  - most of all seeks
    - to achieve a low-risk software development
    - width short development cycles (iterations)
- Every iteration is a complete software development cycle:
  - requirements analysis
  - design
  - implementation
  - testing and documentation
- The goal of an iteration is to perform
  - a tested and workable release of the application at the end of the cycle.

Principles of Agile methods

- Agile methods **prefer face to face contact** instead of written documents.
  - Therefore typical agile teams are located in one a large office.
- At agile methods, the main measure of progress is the **operating software**.
- Rapid adaptation to changing conditions
- **KISS principle:**
  - “Keep it simple, stupid”. Make everything as simple as possible.
- Projects are lead by motivated and trustworthy persons
- Self-organizing teamwork
- Tight, daily communication between the developer and the customer

Agile methods

- Agile methodology is recommended to use
  - in case of small size projects,
  - few (but experienced) developers and
  - often under changing requirements.

- It is not recommended for use
  - with large projects,
  - distributed developments and
  - critical projects.

Extreme Programming...

Extreme Programming

- Perhaps the **best known and widely used agile method**
- The name was coined by Beck in 2000
- **The main idea behind:**
  - pushing recognized good practice to ‘extreme’ levels.
    - such as iterative development
- **For example:**
  - in XP, several new versions of a system may be developed by different programmers,
  - integrated and tested in a day.

Extreme Programming

- In XP, requirements are expressed as **scenarios**
  - called "user stories",
  - which are implemented directly as a series of tasks
- Programmers work in pairs and develop tests for each task before writing the code.
- All tests must be successfully executed when new code is integrated into the system.
- There is a short time gap between releases of the system.

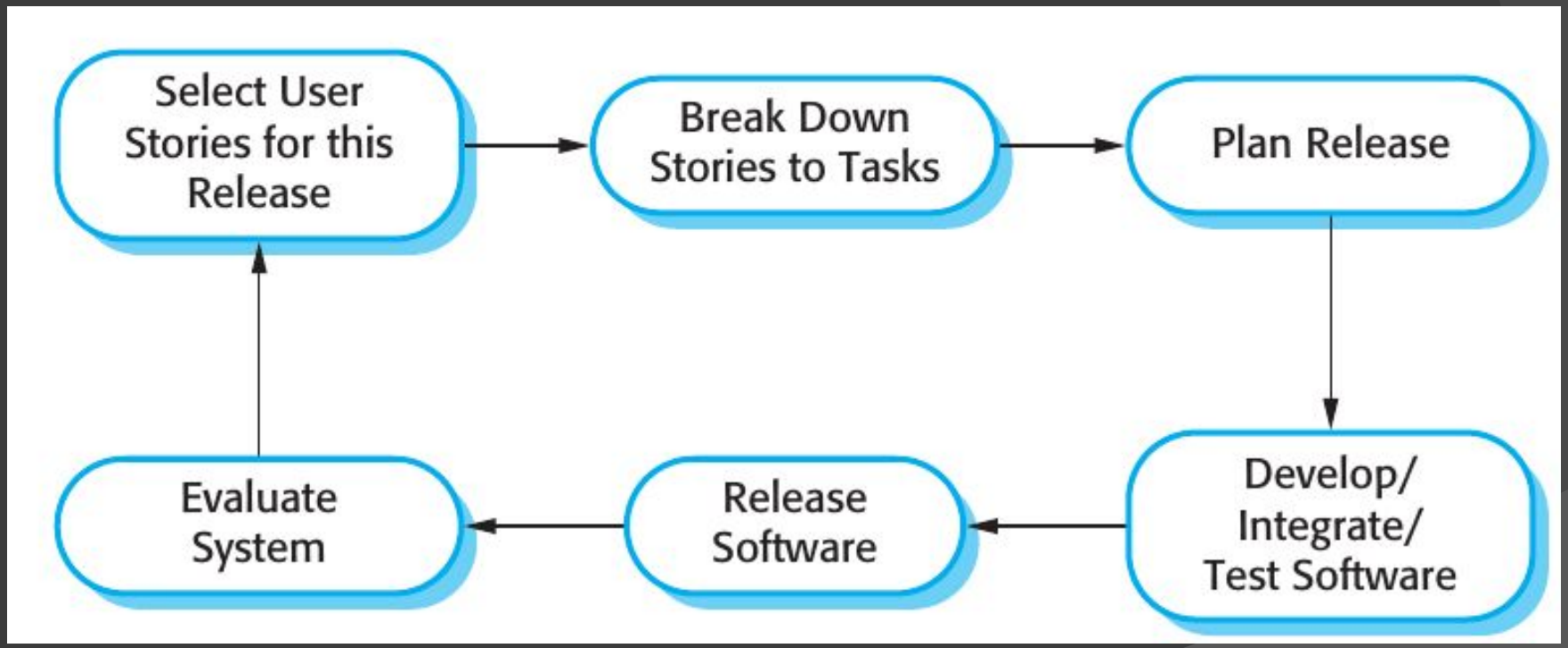
Extreme Programming

- **XP practices which reflect the principles of agile methods:**
  - **Incremental development** is supported through small, frequent releases of the system.
  - Requirements are based on simple customer stories or scenarios
    - that are used as a basis for deciding what functionality should be included in a system increment.
  - **Customer involvement:** supported through the continuous engagement of the customer in the development team.
    - The customer takes part in the development
      - is responsible for defining acceptance tests for the system.

Extreme Programming

- **People are supported through pair programming**,
  - collective ownership of the system code,
  - and a sustainable development process that does not involve excessively long working hours.
- **Change is embraced through regular system releases**
  - test-first development and refactoring to avoid code degeneration
- **Maintaining simplicity** is supported by constant refactoring
  - that improves code quality by using simple designs
    - that do not unnecessarily anticipate future changes to the system.

XP release cycle



Extreme Programming

- Customers are involved in specifying system requirements.
  - Together with the team, a ‘story card’ is developed
    - a story card encapsulates the customer needs
- **The story cards are the main inputs to the XP planning process**
- Once the story cards have been developed
  - 1. the development team breaks these down into tasks
  - 2. estimates the effort and resources required for implementation
  - 3. the customer then prioritizes the stories for implementation
    - choosing those stories that can be used immediately to deliver useful business support.

Extreme Programming

- The aim is to break down functionalities to be implemented in about **two weeks**
  - when the next release of the system is made available to the customer
- When requirements change, the unimplemented stories change or may be discarded.
- If changes are required for a system that has already been delivered, new story cards are developed
  - and again, the customer decides the priority of these changes

Extreme Programming

- **XP takes an 'extreme' approach to incremental development:**
  - New versions of the software may be built several times per day
  - releases are delivered to customers roughly every two weeks
  - Release deadlines are never slipped;
  - If there are development problems
    - the customer is consulted
    - and functionality is removed from the planned release.

Extreme Programming

- A fundamental precept of traditional software engineering is to design for change.
  - to prepare for possible future features
- **XP has discarded this principle**
  - 'designing for change is often wasted effort'
    - It isn't worth taking time to add generality to a program to cope with change.
    - The anticipated changes are often never materialized or they are completely completely different
- **The XP approach:**
  - changes will be implemented when real requirements arise

Extreme Programming

- A general problem with incremental development is that it tends to degrade the software structure,
  - changes to the software become harder and harder to implement
  - Common problems:
    - the code is often duplicated
    - parts of the software are reused in inappropriate ways
    - the overall structure degrades as code is added to the system
- **XP solution: the software should be constantly refactored**
  - the programming team look for possible improvements to the software
    - and implement them immediately.
  - When a team member sees code that can be improved, they make these improvements

Testing in Extreme Programming...

Extreme Programming

- XP emphasizes the importance of program testing
- XP includes an approach to testing
  - that reduces the chances of introducing undiscovered errors into the current version of the system.

The key features of testing in XP are:

- 1. Test-first development
- 2. Incremental test development from scenarios
- 3. User involvement in the test development and validation
- 4. The use of automated testing frameworks.

Test First Development

- Test-first development is one of the most important innovations in XP
  - **we write the tests before you write the code!**
- This means that we can run the test as the code is being written
  - and discover problems during development.
- Writing tests implicitly defines the interface and a specification of the functionality being developed.
- Problems of requirements and interface misunderstandings are reduced.

Test First Development

- The role of the customer in the testing process:
  - to help develop **acceptance tests** for the stories in the next release
    - Acceptance testing is the process where the system is tested using customer data to check that it meets the customer's real needs.
- In XP, acceptance testing is also incremental.
- The customer who is part of the team writes tests as development proceeds.
- All new code is therefore validated to ensure that it is what the customer needs.

Test First Development

- **Test automation is essential for test-first development**
- Tests are written as executable components before the task is implemented.
- These testing components should be
  - standalone,
  - should simulate the submission of input to be tested, and should check that the result meets the output specification.
- An automated test framework is:
  - a system that makes it easy to write executable tests and submit a set of tests for execution.
    - E.g.: JUnit

Pair Programming...

Pair Programming

- Another innovative practice that has been introduced in XP
- **Programmers work in pairs to develop the software**
- They actually sit together at the same workstation to develop the software.
- However, the same pairs do not always program together:
  - Rather, pairs are created dynamically
  - so that all team members work with each other during the development process.

Advantage of pair programming

- **1. It supports the idea of collective ownership and responsibility for the system**
  - The software is owned by the team as a whole
  - Individuals are not held responsible for problems with the code
  - Instead, the team has collective responsibility for resolving these problems.

Advantage of pair programming

- **2. It acts as an informal review process**
  - because each line of code is looked at by at least two people.
  - Much cheaper than code inspections and reviews
  - Does not introduce delays into the development process.

Advantage of pair programming

- **3. It helps support refactoring**
  - refactoring is is a long-term benefit.
  - An individual who practices refactoring may be judged to be less efficient than one who simply carries on developing code.
  - Where pair programming and collective ownership are used, others benefit immediately from the refactoring
    - so they are likely to support the process.

Thank you for your attention!